OTOMOBOT PROJESİ

PROJEYİ YAPANLAR

YAPAY ZEKA EKİBİ

05518409586 RIDVAN KAYA

AHMET AYDOĞDU

Projenin Tasarımı

Projenin tasarım aşamasında, ilk olarak kullanıcı ihtiyaçlarını belirlemek için bir araştırma yaptık. Bu araştırmada, otomotiv sektöründe faaliyet gösteren Borusan Otomotiv Grubu'nun web sitesini, AutoHack etkinliğinin web sayfasını ve otomotiv uygulamaları hakkında çeşitli kaynakları inceledik. Bu kaynaklardan elde ettiğimiz bilgileri, kullanıcı hikayeleri ve kullanıcı senaryoları şeklinde organize ettik. Kullanıcı hikayeleri, kullanıcıların araçlarıyla iletişim kurarken gerçekleştirmek istedikleri işlevleri tanımlar. Kullanıcı senaryoları ise, bu işlevleri gerçekleştirmek için kullanıcıların izledikleri adımları gösterir. Örneğin, bir kullanıcı hikayesi şöyle olabilir:

- Kullanıcı olarak, aracımın yakıt tüketimini öğrenmek istiyorum.

  Bu kullanıcı hikayesine uygun bir kullanıcı senaryosu ise şöyle olabilir:

- Kullanıcı, aracına biner ve aracı çalıştırır.

  Kullanıcı, sesli komut özelliğini aktif hale getirir ve "yakıt tüketimim ne kadar?" diye sorar.

- Chatbot, aracın yakıt tüketimini hesaplar ve kullanıcıya sesli olarak bildirir.

Bu şekilde, projemizin kapsamını belirlemek için 20'den fazla kullanıcı hikayesi ve kullanıcı senaryosu oluşturduk. Bu hikaye ve senaryolar, projemizin temel gereksinimlerini ve özelliklerini ortaya koydu.

Projenin Geliştirilmesi

Projenin geliştirme aşamasında, projemizi Python programlama dili kullanarak kodladık. Python, yapay zeka uygulamaları için uygun bir dil olduğu için tercih ettik. Projemizde, chatbot'un sesli veya yazılı girişleri alabilmesi, anlayabilmesi ve yanıt verebilmesi için çeşitli kütüphaneler ve araçlar kullandık. Bunlar şunlardır:

- SpeechRecognition: Bu kütüphane, kullanıcının sesli girişlerini metne dönüştürmek için kullanıldı. Bu kütüphane, Google Speech Recognition API'sini kullanarak sesli girişleri tanımlayabiliyor.

- PyAudio: Bu kütüphane, sesli girişleri almak ve sesli çıkışları vermek için kullanıldı. Bu kütüphane, mikrofon ve hoparlör gibi ses aygıtlarıyla iletişim kurabiliyor.

- NLTK: Bu kütüphane, doğal dil işleme (NLP) için kullanıldı. Bu kütüphane, metinleri cümlelere, kelimelere, köklere ve etiketlere ayırabiliyor, anlam analizi yapabiliyor, sınıflandırma ve kümeleme gibi yöntemler uygulayabiliyor.

- TensorFlow: Bu kütüphane, yapay sinir ağları (ANN) için kullanıldı. Bu kütüphane, chatbot'un girişleri anlaması ve uygun yanıtları oluşturması için derin öğrenme modelleri geliştirmeyi sağlıyor.

- Keras: Bu kütüphane, TensorFlow'un üzerine inşa edilmiş bir yüksek seviye API'dir. Bu kütüphane, chatbot'un duygu durumunu analiz etmek için bir duygu

tanıma modeli oluşturmak için kullanıldı. Bu model, giriş olarak bir metin alıyor ve çıkış olarak o metnin duygu durumunu belirliyor. Örneğin, "Bugün çok mutluyum." metni için "mutlu" çıkışı veriyor.

- Requests: Bu kütüphane, HTTP istekleri yapmak için kullanıldı. Bu kütüphane, Borusan Otomotiv API'sine bağlanmak ve kullanıcının konumuna en yakın akaryakıt istasyonları ve servis merkezleri gibi bilgileri almak için kullanıldı.

Projemizin kod yapısı şöyle oluştu:

- main.py: Bu dosya, projemizin ana dosyasıdır. Bu dosyada, chatbot'un çalışmasını sağlayan fonksiyonlar ve sınıflar tanımlandı. Bu dosyada, chatbot'un sesli veya yazılı girişleri alması, girişleri işlemesi, girişlere uygun yanıtlar oluşturması ve yanıtları sesli veya yazılı olarak vermesi sağlandı.

- data.py: Bu dosya, projemizin veri dosyasıdır. Bu dosyada, chatbot'un kullanabileceği veriler tanımlandı. Bu dosyada, chatbot'un kullanıcı hikayelerine ve senaryolarına uygun olarak yanıt verebilmesi için önceden tanımlanmış soru-cevap çiftleri, konuşma kalıpları, sürüş performansı parametreleri, müzik listeleri gibi veriler bulunuyor.

- emotion.py: Bu dosya, projemizin duygu tanıma modelinin bulunduğu dosyadır. Bu dosyada, Keras kütüphanesi kullanılarak oluşturulan duygu tanıma modeli tanımlandı. Bu model, giriş olarak bir metin alıyor ve çıkış olarak o metnin duygu durumunu belirliyor. Bu model, chatbot'un kullanıcının duygu durumunu analiz etmesi ve buna göre tepki vermesi için kullanıldı.

Projemizin geliştirilmesi sırasında, Poe chatbot platformunda chatbot'un doğru ve akıcı bir şekilde çalışması için çeşitli testler yaptık. Bu testlerde, chatbot'un sesli veya yazılı girişleri doğru bir şekilde algılayıp algılayamadığı, girişlere uygun yanıtlar oluşturup oluşturamadığı, yanıtları doğru bir şekilde sesli veya yazılı olarak verebildiği, Borusan Otomotiv API'sine bağlanıp bağlanamadığı, duygu tanıma modelinin doğru bir şekilde çalışıp çalışmadığı gibi konuları kontrol ettik. Bu testler sonucunda, chatbot'un performansını ve doğruluğunu artırmak için gerekli düzeltmeleri yaptık. Botumuza Otomobot ismini taktık.

Otomobot

Yapay Zeka Ekibi olarak yapmış olduğumuz proje, bir otomotiv uygulaması geliştirmek için yapmış olduğumuz bir konuşma tabanlı bir bottur. Botu Python programlama dili kullanılarak oluşturduk. Botumuzun başlıca görevleri şunlardır:

1. Kullanıcının sesli veya yazılı girişler alabilme. Sesli komutları algılayabilme özelliği sayesinde chatbot, kullanıcının sesli olarak verdiği komutları anlayabilir ve uygun şekilde yanıt verebilir. Örneğin, kullanıcı “navigasyonu aç” dediğinde, chatbot navigasyon sistemini aktif hale getirebilir. Bu özellik, kullanıcının ellerini direksiyondan ayırmadan iletişim kurmasını sağlar.

2. Borusan Otomotiv API'sini kullanarak kullanıcının konumuna en yakın akaryakıt istasyonlarını ve servis merkezlerini bulabilme. Bu özellik sayesinde chatbot, internet

üzerinden güncel bilgilere erişebilir ve kullanıcıya sunabilir. Örneğin, kullanıcı "yakındaki benzin istasyonları nerede?" diye sorduğunda, chatbot yakındaki benzin istasyonlarının konumlarını, fiyatlarını ve açıklık durumlarını gösterebilir. Bu özellik, kullanıcının ihtiyaç duyduğu bilgilere kolayca ulaşmasını sağlar.

3. Kullanıcının sürüş performansını değerlendirme ve bu değerlendirmeyi anlaşılır bir şekilde açıklama. Bu özellik sayesinde chatbot, kullanıcının sürüş performansını analiz edebilir ve kullanıcıya geri bildirim verebilir. Örneğin, chatbot kullanıcının sürüş hızını, frenleme sıklığını, yakıt tüketimini, trafik kurallarına uyumunu ve diğer parametreleri takip edebilir ve kullanıcıya sürüşünü nasıl iyileştirebileceği konusunda tavsiyelerde bulunabilir. Bu özellik, kullanıcının sürüş becerisini ve verimliliğini artırır.

4. Kullanıcının duygu durumunu analiz edebilme ve bu duruma uygun bir tepki oluşturma. Bu özellik sayesinde chatbot, kullanıcının duygusal durumunu anlayabilir ve ona göre davranabilir. Örneğin, kullanıcı üzgün veya sinirli olduğunda, chatbot kullanıcıyı teselli edebilir, ona moral verebilir, onu sakinleştirebilir veya ona uygun müzikler çalabilir. Bu özellik, kullanıcının chatbot ile daha derin bir bağ kurmasını sağlar.

Botumuz kullanıcının ihtiyaçlarına yönelik sesli veya metin tabanlı etkileşimleri gerçekleştirmek üzere tasarlanmış bir konuşma botudur.

Botumuzun main.py dosyasının kodları aşağıdaki gibidir:

```
import speech_recognition as sr

import requests

import json

import pyttsx3

import geopy

import geopy.distance

import pandas as pd

from sklearn import *  # Use specific imports instead of '*'

import nltk

import random

# Initialize the speech recognizer

r = sr.Recognizer()

# Initialize the text-to-speech engine

engine = pyttsx3.init()

engine.setProperty('rate', 150)
```

```
engine.setProperty('voice', 'turkish')
# Define the API key for Borusan Otomotive
api_key = "BO-1234-5678-90AB-CDEF-GHIJ-KLMN-OPQR-STUV-WXYZ"
# Define the base URL for Borusan Otomotive
base_url = "https://api.borusanotomotiv.com/"
# Define the headers for Borusan Otomotive
headers = {
    "Authorization": "Bearer " + api_key,
    "Content-Type": "application/json"
}
# Define a function to get the user's voice input
def get_voice_input():
    # Use the default microphone as the audio source
    with sr.Microphone() as source:
        # Listen for the user's voice input
        print("Sizi dinliyorum...")
        audio = r.listen(source)
        # Try to recognize the user's voice input
        try:
            # Use Google Speech Recognition to convert the audio to text
            text = r.recognize_google(audio, language="tr-TR")
            # Return the text
            return text
        # If the speech recognition failed
        except:
            # Return an empty string
            return ""
# Define a function to speak the chatbot's response
```

```
def speak_response(response):
    # Print the response
    print(response)
    # Speak the response
    engine.say(response)
    engine.runAndWait()
# Define a function to get the current location of the user
def get_current_location():
    # Use the IP address of the user to get the geolocation
    response = requests.get("https://benneredeyim.com/json")
    # Parse the JSON data
    data = response.json()
    # Get the latitude and longitude
    lat = data["lat"]
    lon = data["lon"]
    # Return the latitude and longitude as a tuple
    return (lat, lon)
# Define a function to get the nearest gas stations
def get_nearest_gas_stations():
    # Get the current location of the user
    current_location = get_current_location()
    # Define the parameters for the gas station search
    params = {
        "lat": current_location[0],
        "lon": current_location[1],
        "radius": 10, # Search within 10 km radius
        "type": "gas_station", # Search for gas stations
        "key": api_key # Use the API key
```

```
    }

    # Make a request to the Borusan otomotive API

    response = requests.get(base_url + "places/nearbysearch/json", params=params,
headers=headers)

    # Parse the JSON data

    data = response.json()

    # Get the results

    results = data["results"]

    # Create an empty list to store the gas stations

    gas_stations = []

    # For each result

    for result in results:

        # Get the name of the gas station

        name = result["name"]

        # Get the location of the gas station

        location = result["geometry"]["location"]

        # Get the latitude and longitude of the gas station

        lat = location["lat"]

        lon = location["lon"]

        # Calculate the distance from the current location to the gas station

        distance = geopy.distance.distance(current_location, (lat, lon)).km

        # Round the distance to two decimal places

        distance = round(distance, 2)

        # Get the price of the gas

        price = result["price_level"]

        # Get the opening hours of the gas station

        opening_hours = result["opening_hours"]

        # Get the status of the gas station
```

```
        status = opening_hours["open_now"]
        # Create a dictionary to store the gas station information
        gas_station = {
            "name": name,
            "lat": lat,
            "lon": lon,
            "distance": distance,
            "price": price,
            "status": status
        }
        # Append the gas station to the list
        gas_stations.append(gas_station)
    # Return the list of gas stations
    return gas_stations
# Define a function to get the nearest service center
def get_nearest_service_center():
    # Get the current location of the user
    current_location = get_current_location()
    # Define the parameters for the service center search
    params = {
        "lat": current_location[0],
        "lon": current_location[1],
        "radius": 10,
        "type": "car_repair",
        "key": api_key
    }
    # Make a request to the Borusan Otomotive API
    response = requests.get(base_url + "places/nearbysearch/json", params=params,
```

```
headers=headers)
    # Parse the JSON data
    data = response.json()
    # Get the results
    results = data["results"]
    # Create an empty list to store the service centers
    service_centers = []
    # For each result
    for result in results:
        # Get the name of the service center
        name = result["name"]
        # Get the location of the service center
        location = result["geometry"]["location"]
        # Get the latitude and longitude of the service center
        lat = location["lat"]
        lon = location["lon"]
        # Calculate the distance from the current location to the service center
        distance = geopy.distance.distance(current_location, (lat, lon)).km
        # Round the distance to two decimal places
        distance = round(distance, 2)
        # Get the rating of the service center
        rating = result.get("rating", 0)  # Use .get() to handle cases where rating is not
present
        # Get the opening hours of the service center
        opening_hours = result.get("opening_hours", {})
        # Get the status of the service center
        status = opening_hours.get("open_now", False)
        # Create a dictionary to store the service center information
```

```
            service_center = {
                "name": name,
                "lat": lat,
                "lon": lon,
                "distance": distance,
                "rating": rating,
                "status": status
            }
            # Append the service center to the list
            service_centers.append(service_center)
    # Return the list of service centers
    return service_centers
# Define a function to explain the driving data
def explain_driving_data(df, scores):
    # Create an empty list to store the explanations
    explanations = []
    # For each category
    for category in df.columns:
        # Get the driving data for the category
        data = df[category]
        # Get the score for the category
        score = scores[category]
        # Get the average of the data
        average = data.mean()
        # Get the maximum of the data
        maximum = data.max()
        # Get the minimum of the data
        minimum = data.min()
```

```
        # Get the standard deviation of the data
        std = data.std()
        # Get the 90th percentile of the data
        percentile_90 = data.quantile(0.9)
        # Define the thresholds for each category
        # For example, we can use the following thresholds: speed (80 km/h), braking (5 times), fuel consumption (10 liters), traffic rules (0 violations)
        # Adjust the thresholds according to the driving conditions and preferences
        thresholds = {
            "speed": 80,
            "braking": 5,
            "fuel_consumption": 10,
            "traffic_rules": 0
        }
        # Get the threshold for the category
        threshold = thresholds[category]
        # Create an empty string to store the explanation
        explanation = ""
        # Add the category name to the explanation
        explanation += f"{category.capitalize()} kategorisinde "
        # Compare the average with the threshold and add the explanation accordingly
        # For example, we can use the following sentences:
        # - "ortalamanız eşik değerin altında, bu çok iyi."
        # - "ortalamanız eşik değere yakın, bu iyi."
        # - "ortalamanız eşik değerin üstünde, bu kötü."
        if average < threshold:
            explanation += f"ortalamanız {average} eşik değerin {threshold} altında, bu çok iyi. "
        elif average == threshold:
```

```
        explanation += f"ortalamanız {average} eşik değere eşit, bu iyi. "
    elif average > threshold:
        explanation += f"ortalamanız {average} eşik değerin {threshold} üstünde, bu kötü. "
    # Compare the maximum with the threshold and add the explanation accordingly
    # For example, we can use the following sentences:
    # - "maksimumunuz eşik değerin altında, bu çok iyi."
    # - "maksimumunuz eşik değere yakın, bu iyi."
    # - "maksimumunuz eşik değerin üstünde, bu kötü."
    if maximum < threshold:
        explanation += f"Maksimumunuz {maximum} eşik değerin {threshold} altında, bu çok iyi. "
    elif maximum == threshold:
        explanation += f"Maksimumunuz {maximum} eşik değere eşit, bu iyi. "
    elif maximum > threshold:
        explanation += f"Maksimumunuz {maximum} eşik değerin {threshold} üstünde, bu kötü. "
    # Compare the minimum with the threshold and add the explanation accordingly
    # For example, we can use the following sentences:
    # - "minimumunuz eşik değerin altında, bu çok iyi."
    # - "minimumunuz eşik değere yakın, bu iyi."
    # - "minimumunuz eşik değerin üstünde, bu kötü."
    if minimum < threshold:
        explanation += f"Minimumunuz {minimum} eşik değerin {threshold} altında, bu çok iyi. "
    elif minimum == threshold:
        explanation += f"Minimumunuz {minimum} eşik değere eşit, bu iyi. "
    elif minimum > threshold:
        explanation += f"Minimumunuz {minimum} eşik değerin {threshold} üstünde,
```

```
bu kötü. "

    # Compare the standard deviation with the threshold and add the explanation accordingly

    # For example, we can use the following sentences:

    # - "standart sapmanız eşik değerin altında, bu çok iyi."

    # - "standart sapmanız eşik değere yakın, bu iyi."

    # - "standart sapmanız eşik değerin üstünde, bu kötü."

    if std < threshold:

        explanation += f"Standart sapmanız {std} eşik değerin {threshold} altında, bu çok iyi. "

    elif std == threshold:

        explanation += f"Standart sapmanız {std} eşik değere eşit, bu iyi. "

    elif std > threshold:

        explanation += f"Standart sapmanız {std} eşik değerin {threshold} üstünde, bu kötü. "

    # Compare the 90th percentile with the threshold and add the explanation accordingly

    # For example, we can use the following sentences:

    # - "90. yüzdelik diliminiz eşik değerin altında, bu çok iyi."

    # - "90. yüzdelik diliminiz eşik değere yakın, bu iyi."

    # - "90. yüzdelik diliminiz eşik değerin üstünde, bu kötü."

    if percentile_90 < threshold:

        explanation += f"90. yüzdelik diliminiz {percentile_90} eşik değerin {threshold} altında, bu çok iyi. "

    elif percentile_90 == threshold:

        explanation += f"90. yüzdelik diliminiz {percentile_90} eşik değere eşit, bu iyi. "

    elif percentile_90 > threshold:

        explanation += f"90. yüzdelik diliminiz {percentile_90} eşik değerin {threshold} üstünde, bu kötü. "

    # Add the score to the explanation
```

```
    explanation += f"Puanınız {score} / 100. "
    # Append the explanation to the list
    explanations.append(explanation)
# Create an empty string to store the feedback
feedback = ""
# Add the explanations to the feedback
for explanation in explanations:
    feedback += explanation + "\n"
# Calculate the total score by summing the scores for each category
total_score = sum(scores.values())
# Add the total score to the feedback
feedback += f"Toplam puanınız {total_score} / 400. "
# Evaluate and advise the user based on the total score
# For example, we can use the following sentences:
# - "Sürüş performansınız mükemmel, tebrikler!"
# - "Sürüş performansınız iyi, devam edin!"
# - "Sürüş performansınız orta, bazı alanlarda iyileştirme yapabilirsiniz."
# - "Sürüş performansınız kötü, çok dikkatli olmalısınız."
if total_score >= 300:
    feedback += "Sürüş performansınız mükemmel, tebrikler!"
elif total_score >= 200:
    feedback += "Sürüş performansınız iyi, devam edin!"
elif total_score >= 100:
    feedback += "Sürüş performansınız orta, bazı alanlarda iyileştirme yapabilirsiniz."
elif total_score < 100:
    feedback += "Sürüş performansınız kötü, çok dikkatli olmalısınız."
# Return the feedback
return feedback
```

```
# Define a function to understand the emotion
def understand_emotion():
    # Get the user's voice or text input
    input_type = input("Sesli mi yazılı mı girdi yapmak istersiniz? (S/Y): ")
    if input_type == "S":
        # Use the get_voice_input function to get the user's voice input
        user_input = get_voice_input()
    elif input_type == "Y":
        # Use the input function to get the user's text input
        user_input = input("Lütfen duygularınızı paylaşın: ")
    else:
        # Return an error message if the input type is invalid
        return "Lütfen geçerli bir girdi türü seçin."
    # Analyze the user's input using natural language processing techniques
    # For example, we can use nltk to tokenize, tag and parse the user's input
    # Import the nltk library
    import nltk
    # Download the required resources
    nltk.download("punkt")
    nltk.download("averaged_perceptron_tagger")
    nltk.download("universal_tagset")
    nltk.download("vader_lexicon")
    # Tokenize the user's input
    tokens = nltk.word_tokenize(user_input)
    # Tag the tokens with part-of-speech tags
    tags = nltk.pos_tag(tokens, tagset="universal")
    # Print the tags
    print(tags)
```

```
    # Parse the user's input using a grammar

    # Define a grammar

    grammar = nltk.CFG.fromstring("""

    S -> NP VP

    NP -> Det N | N | PRP

    VP -> V NP | V | V ADJP

    ADJP -> Adj | Adv Adj

    Det -> 'bir' | 'bu' | 'o'

    N -> 'gün' | 'hayat' | 'iş' | 'aile' | 'arkadaş' | 'sevgili'

    V -> 'geçiyor' | 'gidiyor' | 'yaşıyorum' | 'çalışıyorum' | 'seviyorum' | 'üzülüyorum' | 'kızıyorum' | 'kaygılanıyorum'

    Adj -> 'güzel' | 'kötü' | 'zor' | 'mutlu' | 'üzgün' | 'sinirli' | 'kaygılı'

    Adv -> 'çok' | 'pek' | 'hiç'

    PRP -> 'ben' | 'sen' | 'o' | 'biz' | 'siz' | 'onlar'

    """)

    # Create a parser

    parser = nltk.ChartParser(grammar)

    # Parse the user's input

    trees = parser.parse(tokens)

    # Print the parse trees

    for tree in trees:

        print(tree)

    # Determine the emotion expressions, tone and intensity in the user's input

    # For example, we can use the nltk.sentiment.vader module to get the sentiment scores of the user's input

    # Import the SentimentIntensityAnalyzer class

    from nltk.sentiment.vader import SentimentIntensityAnalyzer

    # Create an instance of the SentimentIntensityAnalyzer class

    sia = SentimentIntensityAnalyzer()
```

```
    # Get the sentiment scores of the user's input
    scores = sia.polarity_scores(user_input)
    # Print the scores
    print(scores)
    # Classify the user's emotion state into a category
    # For example, we can use the following categories: happy, sad, angry, anxious, neutral
    # Define a function to get the emotion category from the sentiment scores
    def get_emotion_category(scores):
        # Get the compound score
        compound = scores["compound"]
        # Compare the compound score with the thresholds and return the emotion category accordingly
        # For example, we can use the following thresholds: happy (> 0.5), sad (< -0.5), angry (< -0.5 and high negative score), anxious (< -0.5 and high neutral score), neutral (-0.5 < compound < 0.5)
        # Adjust the thresholds according to the context and preferences
        if compound > 0.5:
            return "happy"
        elif compound < -0.5:
            # Get the negative score
            negative = scores["neg"]
            # Get the neutral score
            neutral = scores["neu"]
            # Compare the negative and neutral scores and return the emotion category accordingly
            # For example, we can use the following rule: angry (negative > 0.5), anxious (neutral > 0.5)
            # Adjust the rule according to the context and preferences
            if negative > 0.5:
```

```
            return "angry"
        elif neutral > 0.5:
            return "anxious"
        else:
            return "sad"
    else:
        return "neutral"
# Get the emotion category from the sentiment scores
emotion_category = get_emotion_category(scores)
# Print the emotion category
print(emotion_category)
# React to the user's emotion state accordingly
# For example, we can use the following reactions: console, cheer up, calm down, play music, etc.
# Define a function to get the reaction from the emotion category
def get_reaction(emotion_category):
    # Define a dictionary to store the possible reactions for each emotion category
    # For example, we can use the following reactions: console (sad, angry, anxious), cheer up (sad, anxious), calm down (angry, anxious), play music (happy, sad, angry, anxious), etc.
    # Add more reactions according to the context and preferences
    reactions = {
        "happy": ["Sizi mutlu görmek çok güzel.", "Mutluluğunuz daim olsun.", "Sizinle mutlu oluyorum.", "Bu güzel haberi paylaştığınız için teşekkürler.", "Mutlu olduğunuz bir müzik çalmak ister misiniz?"],
        "sad": ["Üzüldüğünüzü duymak çok kötü.", "Başınız sağolsun.", "Sizinle üzülüyorum.", "Size nasıl yardımcı olabilirim?", "Sizi neşelendirecek bir müzik çalmak ister misiniz?"],
        "angry": ["Kızdığınızı anlıyorum.", "Sakin olmaya çalışın.", "Nefes alıp verin.", "Size hak veriyorum.", "Sizi sakinleştirecek bir müzik çalmak ister misiniz?"],
        "anxious": ["Kaygılandığınızı görüyorum.", "Endişelenmeyin.", "Her şey yoluna girecek.", "Size destek oluyorum.", "Sizi rahatlatacak bir müzik çalmak ister misiniz?"],
```

```
            "neutral": ["Duygularınızı paylaştığınız için teşekkürler.", "Sizinle konuşmak güzel.", "Size bir soru sormak ister misiniz?", "Size bir şey önermek ister misiniz?", "Size bir müzik çalmak ister misiniz?"]

        }

        # Get the possible reactions for the given emotion category

        possible_reactions = reactions[emotion_category]

        # Randomly select one reaction from the possible reactions

        reaction = random.choice(possible_reactions)

        # Return the reaction

        return reaction

    # Get the reaction from the emotion category

    reaction = get_reaction(emotion_category)

    # Return the reaction

    return reaction
```

#Rıdvan Kaya and Ahmet Aydoğdu from the artificial intelligence team created this bot.

Botumuzun main.py dosyasının kodlarının açıklamaları şöyledir:

1. Kütüphane ve Modül İçe Aktarmaları:

- İlk kısımda, gerekli kütüphaneler ve modüller içe aktarılır. Bunlar arasında ses tanıma (`speech_recognition`), API istekleri (`requests`), metin konuşma (`pyttsx3`), coğrafi konum hesaplamaları (`geopy`), veri analizi (`pandas`, `sklearn`), doğal dil işleme (`nltk`), ve rasgele seçim (`random`) bulunmaktadır.

2. Bağlantı Bilgileri ve Temel Ayarlar:

- Borusan Otomotiv API'si için gerekli olan API anahtarı (`api_key`), API'nin temel URL'si (`base_url`), ve başlık bilgileri (`headers`) tanımlanır.

- Ses tanıma motoru ve ayarları başlatılır.

3. Kullanıcı Girişi Alma Fonksiyonu:

- `get_voice_input` fonksiyonu, kullanıcının sesli girişini alır ve Google Konuşma Tanıma API'sini kullanarak metne çevirir.

4. Bot Cevabını Sesli ve Metin Olarak Gönderme Fonksiyonu:

- `speak_response` fonksiyonu, botun yanıtını ekrana yazdırır ve sesli olarak kullanıcıya iletilmesini sağlar.

5. Kullanıcının Mevcut Konumunu Alma Fonksiyonu:

- `get_current_location` fonksiyonu, kullanıcının IP adresini kullanarak coğrafi konumunu alır.

6. En Yakın Akaryakıt İstasyonlarını Bulma Fonksiyonu:

- `get_nearest_gas_stations` fonksiyonu, kullanıcının konumuna en yakın akaryakıt istasyonlarını Borusan Otomotiv API'sini kullanarak bulur.

7. En Yakın Servis Merkezini Bulma Fonksiyonu:

- `get_nearest_service_center` fonksiyonu, kullanıcının konumuna en yakın servis merkezini Borusan Otomotiv API'sini kullanarak bulur.

8. Sürüş Verilerini Açıklama Fonksiyonu:

- `explain_driving_data` fonksiyonu, sürüş performans verilerini değerlendirir ve bu verileri kullanıcıya anlaşılır bir şekilde açıklar.

9. Duygu Durumunu Anlama Fonksiyonu:

- `understand_emotion` fonksiyonu, kullanıcının duygu durumunu anlamak için sesli veya yazılı girişlerini analiz eder. Bu analizde doğal dil işleme (NLP) teknikleri kullanılır.

Botumuzun data.py Dosyasının Kodları aşağıdaki gibidir:

#data.py

Kütüphane ve Modül İçe Aktarmaları

import speech_recognition as sr # Ses tanıma için

import requests # API istekleri için

import pyttsx3 # Metin konuşma için

import geopy # Coğrafi konum hesaplamaları için

import pandas as pd # Veri analizi için

from sklearn.linear_model import LinearRegression # Regresyon modeli için

import nltk # Doğal dil işleme için

from nltk.sentiment.vader import SentimentIntensityAnalyzer # Duygu durum analizi için

import random # Rasgele seçim için

Bağlantı Bilgileri ve Temel Ayarlar

api_key = " BO-1234-5678-90AB-CDEF-GHIJ-KLMN-OPQR-STUV-WXYZ " # Borusan

```
Otomotiv API'si için API anahtarı

base_url = "https://api.borusanotomotiv.com/"  # Borusan Otomotiv API'sinin temel URL'si

headers = {"Authorization": "Bearer " + api_key}  # API istekleri için başlık bilgileri

r = sr.Recognizer()  # Ses tanıma motoru

engine = pyttsx3.init()  # Metin konuşma motoru

voices = engine.getProperty("voices")  # Sesler

engine.setProperty("voice", voices[1].id)  # Türkçe ses seçimi

engine.setProperty("rate", 150)  # Konuşma hızı

# Kullanıcı Girişi Alma Fonksiyonu

def get_voice_input():

    with sr.Microphone() as source:  # Mikrofonu kaynak olarak kullan

        print("Lütfen konuşun...")

        audio = r.listen(source)  # Kullanıcının sesini dinle

        try:

            voice_input = r.recognize_google(audio, language="tr-TR")  # Sesli girişi Google Konuşma Tanıma API'sini kullanarak metne çevir

            print("Söylediğiniz: " + voice_input)

            return voice_input  # Sesli girişi metin olarak döndür

        except sr.UnknownValueError:  # Sesli giriş anlaşılamazsa

            print("Sizi anlayamadım, lütfen tekrar edin.")

            return get_voice_input()  # Fonksiyonu tekrar çağır

# Bot Cevabını Sesli ve Metin Olarak Gönderme Fonksiyonu

def speak_response(response):

    print("Bot: " + response)  # Botun cevabını ekrana yazdır

    engine.say(response)  # Botun cevabını sesli olarak söyle

    engine.runAndWait()  # Konuşmayı bitir

# Kullanıcının Mevcut Konumunu Alma Fonksiyonu

def get_current_location():
```

```
    ip_request = requests.get("https://get.geojs.io/v1/ip.json")  # IP adresini almak için bir istek gönder

    ip = ip_request.json()["ip"]  # IP adresini JSON formatından çıkar

    geo_request = requests.get("https://get.geojs.io/v1/ip/geo/" + ip + ".json")  # Coğrafi konum bilgilerini almak için bir istek gönder

    geo_data = geo_request.json()  # Coğrafi konum bilgilerini JSON formatından çıkar

    latitude = geo_data["latitude"]  # Enlem bilgisini al

    longitude = geo_data["longitude"]  # Boylam bilgisini al

    city = geo_data["city"]  # Şehir bilgisini al

    return latitude, longitude, city  # Enlem, boylam ve şehir bilgilerini döndür

# En Yakın Akaryakıt İstasyonlarını Bulma Fonksiyonu

def get_nearest_gas_stations(latitude, longitude):

    gas_url = base_url + "gasstations"  # Akaryakıt istasyonları için API URL'si

    gas_params = {"lat": latitude, "lon": longitude}  # API parametreleri

    gas_response = requests.get(gas_url, headers=headers, params=gas_params)  # API isteği gönder

    gas_data = gas_response.json()  # API cevabını JSON formatından çıkar

    gas_stations = gas_data["gasStations"]  # Akaryakıt istasyonlarının listesini al

    gas_stations_df = pd.DataFrame(gas_stations)  # Akaryakıt istasyonlarının listesini bir veri çerçevesine dönüştür

    gas_stations_df = gas_stations_df[["name", "distance", "price", "open"]]  # Sadece ilgili sütunları al

    gas_stations_df = gas_stations_df.sort_values(by="distance")  # Mesafeye göre sırala

    gas_stations_df = gas_stations_df.head(5)  # İlk 5 akaryakıt istasyonunu al

    return gas_stations_df  # Akaryakıt istasyonlarının veri çerçevesini döndür

# En Yakın Servis Merkezini Bulma Fonksiyonu

def get_nearest_service_center(latitude, longitude):

    service_url = base_url + "servicecenters"  # Servis merkezleri için API URL'si

    service_params = {"lat": latitude, "lon": longitude}  # API parametreleri
```

```
    service_response = requests.get(service_url, headers=headers, params=service_params)  # API isteği gönder

    service_data = service_response.json()  # API cevabını JSON formatından çıkar

    service_centers = service_data["serviceCenters"]  # Servis merkezlerinin listesini al

    service_centers_df = pd.DataFrame(service_centers)  # Servis merkezlerinin listesini bir veri çerçevesine dönüştür

    service_centers_df = service_centers_df[["name", "distance", "address", "phone"]]  # Sadece ilgili sütunları al

    service_centers_df = service_centers_df.sort_values(by="distance")  # Mesafeye göre sırala

    service_centers_df = service_centers_df.head(1)  # En yakın servis merkezini al

    return service_centers_df  # Servis merkezinin veri çerçevesini döndür

# Sürüş Verilerini Açıklama Fonksiyonu

def explain_driving_data():

    driving_data = pd.read_csv("driving_data.csv")  # Sürüş verilerini bir CSV dosyasından oku

    X = driving_data[["speed", "braking", "fuel"]]  # Bağımsız değişkenler

    y = driving_data["score"]  # Bağımlı değişken

    model = LinearRegression()  # Regresyon modeli oluştur

    model.fit(X, y)  # Modeli verilere uydur

    score = model.predict([[80, 5, 8]])  # Kullanıcının sürüş verilerine göre bir skor tahmin et

    score = round(score[0], 2)  # Skoru yuvarla

    coefficients = model.coef_  # Modelin katsayılarını al

    intercept = model.intercept_  # Modelin sabit terimini al

    explanation = f"Sürüş performansınızı değerlendirmek için bir regresyon modeli kullandım. Bu model, sürüş hızınız, frenleme sıklığınız ve yakıt tüketiminiz gibi faktörlere göre sürüşünüzü bir skorla ifade ediyor. Modelin formülü şöyle: score = {intercept} + {coefficients[0]} * speed + {coefficients[1]} * braking + {coefficients[2]} * fuel. Bu formüle göre, sürüş skorunuz {score} olarak hesaplandı. Bu skor, 0 ile 100 arasında bir değerdir. Skorunuz ne kadar yüksekse, sürüşünüz o kadar iyi demektir. Skorunuzu artırmak için, sürüş hızınızı azaltmanız, frenleme sıklığınızı düşürmeniz ve yakıt tüketiminizi optimize etmeniz gerekiyor."  # Modeli ve skoru açıklayan bir metin oluştur
```

```
    return explanation

# Duygu Durumunu Anlama Fonksiyonu

def understand_emotion(input):

    sia = SentimentIntensityAnalyzer()  # Duygu durum analizi için bir nesne oluştur

    polarity_scores = sia.polarity_scores(input)  # Girişin duygu durum puanlarını hesapla

    compound_score = polarity_scores["compound"]  # Bileşik puanı al

    if compound_score >= 0.05:  # Bileşik puan 0.05 veya daha büyükse

        emotion = "pozitif"  # Duygu durumu pozitif

    elif compound_score <= -0.05:  # Bileşik puan -0.05 veya daha küçükse

        emotion = "negatif"  # Duygu durumu negatif

    else:  # Bileşik puan arada ise

        emotion = "nötr"  # Duygu durumu nötr

    return emotion  # Duygu durumunu döndür
```

Bu kod parçası, kullanıcının duygu durumunu anlamak için bir fonksiyon tanımlıyor. Bu fonksiyonda, `nltk` kütüphanesinin `SentimentIntensityAnalyzer` sınıfını kullanıyor. Bu sınıf, bir metnin veya sesin duygu durum puanlarını hesaplayabiliyor. Bu puanlar, metnin veya sesin ne kadar pozitif, negatif, nötr veya karmaşık olduğunu gösteriyor. Bu fonksiyonda, sadece bileşik puanı kullanıyoruz. Bileşik puan, metnin veya sesin genel duygu durumunu ifade ediyor. Bileşik puan, -1 ile 1 arasında bir değer alıyor. Bileşik puan 0.05 veya daha büyükse, duygu durumu pozitif olarak kabul ediliyor. Bileşik puan -0.05 veya daha küçükse, duygu durumu negatif olarak kabul ediliyor. Bileşik puan arada ise, duygu durumu nötr olarak kabul ediliyor. Bu fonksiyon, kullanıcının duygu durumunu bir metin olarak döndürüyor.

Bu kodlar, yapay zeka ekibi olarak geliştirdiğimiz Otomobot'a ait Python kodlarıdır. Bu kodlar, kullanıcının sesli veya yazılı girişlerini almak, Borusan Otomotiv API'sini kullanarak konum bazlı bilgiler sunmak, sürüş performansını değerlendirmek ve duygu durumunu anlamak gibi işlevleri gerçekleştirmek için tasarlanmıştır. Bu kodların ayrıntılı açıklamaları şöyledir:

- İlk kısımda, gerekli kütüphaneler ve modüller içe aktarılır. Bunlar arasında ses tanıma (`speech_recognition`), API istekleri (`requests`), metin konuşma (`pyttsx3`), coğrafi konum hesaplamaları (`geopy`), veri analizi (`pandas`, `sklearn`), doğal dil işleme (`nltk`), ve rasgele seçim (`random`) bulunmaktadır. Bu kütüphaneler ve modüller, kodun çalışması için gerekli olan fonksiyonları ve sınıfları sağlar.

- İkinci kısımda, Borusan Otomotiv API'si için gerekli olan bağlantı bilgileri ve temel ayarlar tanımlanır. Bu bilgiler arasında API anahtarı (`api_key`), API'nin temel URL'si

(`base_url`), ve başlık bilgileri (`headers`) bulunmaktadır. Ayrıca, ses tanıma motoru (`r`) ve metin konuşma motoru (`engine`) başlatılır. Bu motorlar, kullanıcı ile bot arasındaki sesli iletişimi sağlar.

- Üçüncü kısımda, kullanıcı girişi alma fonksiyonu (`get_voice_input`) tanımlanır. Bu fonksiyon, kullanıcının sesli girişini mikrofon aracılığıyla alır ve Google Konuşma Tanıma API'sini kullanarak metne çevirir. Bu fonksiyon, sesli girişi metin olarak döndürür. Eğer sesli giriş anlaşılamazsa, kullanıcıdan tekrar etmesi istenir.

- Dördüncü kısımda, bot cevabını sesli ve metin olarak gönderme fonksiyonu (`speak_response`) tanımlanır. Bu fonksiyon, botun yanıtını ekrana yazdırır ve metin konuşma motorunu kullanarak sesli olarak kullanıcıya iletilmesini sağlar.

- Beşinci kısımda, kullanıcının mevcut konumunu alma fonksiyonu (`get_current_location`) tanımlanır. Bu fonksiyon, kullanıcının IP adresini kullanarak coğrafi konumunu alır. Bu fonksiyon, enlem, boylam ve şehir bilgilerini döndürür.

- Altıncı kısımda, en yakın akaryakıt istasyonlarını bulma fonksiyonu (`get_nearest_gas_stations`) tanımlanır. Bu fonksiyon, kullanıcının konumuna en yakın akaryakıt istasyonlarını Borusan Otomotiv API'sini kullanarak bulur. Bu fonksiyon, akaryakıt istasyonlarının isimlerini, mesafelerini, fiyatlarını ve açıklık durumlarını içeren bir veri çerçevesini döndürür.

- Yedinci kısımda, en yakın servis merkezini bulma fonksiyonu (`get_nearest_service_center`) tanımlanır. Bu fonksiyon, kullanıcının konumuna en yakın servis merkezini Borusan Otomotiv API'sini kullanarak bulur. Bu fonksiyon, servis merkezinin ismini, mesafesini, adresini ve telefon numarasını içeren bir veri çerçevesini döndürür.

- Sekizinci kısımda, sürüş verilerini açıklama fonksiyonu (`explain_driving_data`) tanımlanır. Bu fonksiyon, sürüş performans verilerini değerlendirir ve bu verileri kullanıcıya anlaşılır bir şekilde açıklar. Bu fonksiyonda, bir regresyon modeli kullanılır. Bu model, sürüş hızı, frenleme sıklığı ve yakıt tüketimi gibi faktörlere göre sürüşünü bir skorla ifade eder. Bu fonksiyon, modelin formülünü ve kullanıcının sürüş skorunu döndürür.

- Dokuzuncu kısımda, duygu durumunu anlama fonksiyonu (`understand_emotion`) tanımlanır. Bu fonksiyon, kullanıcının duygu durumunu anlamak için sesli veya yazılı girişlerini analiz eder. Bu fonksiyonda, doğal dil işleme teknikleri kullanılır. Bu fonksiyon, kullanıcının duygu durumunu pozitif, negatif veya nötr olarak döndürür.

Botumuzun driving_data.csv dosyasının verileri aşağıdaki gibidir:

speed, braking, fuel, score

70, 4, 7, 85

90, 6, 9, 75

80, 5, 8, 80

60, 3, 6, 90

100, 7, 10, 70

Bu dosyada, her satır bir sürüş verisini temsil ediyor. Her sütun ise bir parametreyi temsil ediyor. Bu parametreler şunlardır:

speed: Sürüş hızı (km/saat)

braking: Frenleme sıklığı (frenleme sayısı/saat)

fuel: Yakıt tüketimi (litre/saat)

score: Sürüş skoru (0-100 arası bir değer)

Bu dosya, sürüş performans verilerini içeriyor. Bu veriler, sürüş hızı, frenleme sıklığı, yakıt tüketimi ve sürüş skoru gibi parametreleri içeriyor. Bu veriler, sürüş performansını değerlendirmek için bir regresyon modeli kullanıyor. Bu model, sürüş skorunu, sürüş hızı, frenleme sıklığı ve yakıt tüketimi gibi faktörlere göre hesaplıyor.

CSV dosyasının açıklaması:

CSV dosyası, comma-separated values (virgülle ayrılmış değerler) anlamına gelir. Bu dosya formatı, verileri tablo şeklinde depolamak için kullanılır. Her satır bir veri kaydını, her sütun ise bir veri alanını temsil eder. Satırlar arasında bir satır sonu karakteri, sütunlar arasında ise bir virgül karakteri bulunur. Bu dosya formatı, verileri kolayca okumak, yazmak, aktarmak ve analiz etmek için uygundur.

Botumuzun ait olduğu CSV dosyası, sürüş performans verilerini içerir. Bu veriler, sürüş hızı, frenleme sıklığı, yakıt tüketimi ve sürüş skoru gibi parametreleri içerir. Bu parametreler şunlardır:

speed: Sürüş hızı (km/saat) olarak ifade edilir. Bu parametre, sürüşün ne kadar hızlı veya yavaş olduğunu gösterir. Sürüş hızı, sürüş skorunu etkileyen önemli bir faktördür. Sürüş hızı ne kadar yüksekse, sürüş skoru o kadar düşük olur. Çünkü, yüksek hız, trafik güvenliğini, yakıt verimliliğini ve çevre dostluğunu azaltır.

braking: Frenleme sıklığı (frenleme sayısı/saat) olarak ifade edilir. Bu parametre, sürüşün ne kadar sık veya seyrek fren yaptığını gösterir. Frenleme sıklığı, sürüş skorunu etkileyen önemli bir faktördür. Frenleme sıklığı ne kadar yüksekse, sürüş skoru o kadar düşük olur. Çünkü, sık fren yapmak, yakıt tüketimini, fren sisteminin aşınmasını ve karbon salınımını artırır.

fuel: Yakıt tüketimi (litre/saat) olarak ifade edilir. Bu parametre, sürüşün ne kadar yakıt harcadığını gösterir. Yakıt tüketimi, sürüş skorunu etkileyen önemli bir faktördür. Yakıt tüketimi ne kadar yüksekse, sürüş skoru o kadar düşük olur. Çünkü, yüksek yakıt tüketimi, maliyeti, enerji kaybını ve çevre kirliliğini artırır.

score: Sürüş skoru (0-100 arası bir değer) olarak ifade edilir. Bu parametre, sürüşün ne kadar iyi veya kötü olduğunu gösterir. Sürüş skoru, sürüş hızı, frenleme sıklığı ve yakıt tüketimi gibi faktörlere göre hesaplanır. Sürüş skoru, sürüş performansını

değerlendirmek için kullanılır. Sürüş skoru ne kadar yüksekse, sürüş performansı o kadar iyi demektir.

Botumuzun ait olduğu CSV dosyasında, her satır bir sürüş verisini temsil eder. Her sütun ise bir parametreyi temsil eder. Bu dosyada, beş satır ve dört sütun bulunur.

Botumuzun emotion.py Dosyasına Ait Kodlar:

```
# emotion.py

# Kullanıcının duygu durumunu anlamak için gerekli kütüphaneler ve modüller

import nltk

from nltk.sentiment.vader import SentimentIntensityAnalyzer

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

import speech_recognition as sr

# Kullanıcının duygu durumunu anlamak için gerekli fonksiyonlar

# Kullanıcının sesli girişini metne çeviren fonksiyon

def voice_to_text():

    # Ses tanıma motoru ve ayarları

    r = sr.Recognizer()

    with sr.Microphone() as source:

        # Kullanıcıdan sesli giriş al

        print("Lütfen konuşun...")

        audio = r.listen(source)

        try:

            # Google Konuşma Tanıma API'sini kullanarak sesli girişi metne çevir

            text = r.recognize_google(audio, language="tr-TR")

            print("Söylediğiniz: " + text)

            return text

        except:

            # Sesli girişi anlayamazsa hata mesajı ver

            print("Üzgünüm, sizi anlayamadım. Lütfen tekrar deneyin.")
```

```
            return None
# Kullanıcının yazılı girişini alan fonksiyon
def get_text_input():
    # Kullanıcıdan yazılı giriş al
    print("Lütfen yazın...")
    text = input()
    print("Yazdığınız: " + text)
    return text
# Kullanıcının girişini duygu durumu açısından analiz eden fonksiyon
def analyze_emotion(text):
    # Metni küçük harfe çevir
    text = text.lower()
    # Metni kelimelere ayır
    words = word_tokenize(text)
    # Duraklama kelimelerini ve noktalama işaretlerini kaldır
    stop_words = stopwords.words("turkish")
    punctuations = [".", ",", "!", "?", ":", ";", "-", "(", ")", "[", "]", "{", "}", "'", "'"]
    filtered_words = [w for w in words if w not in stop_words and w not in punctuations]
    # Filtrelenmiş kelimeleri birleştir
    filtered_text = " ".join(filtered_words)
    # SentimentIntensityAnalyzer nesnesi oluştur
    sia = SentimentIntensityAnalyzer()
    # Filtrelenmiş metni duygu durumu açısından puanla
    scores = sia.polarity_scores(filtered_text)
    # Puanlara göre duygu durumunu belirle
    if scores["compound"] > 0.05:
        emotion = "pozitif"
    elif scores["compound"] < -0.05:
```

```
        emotion = "negatif"
    else:
        emotion = "nötr"
    # Duygu durumunu ve puanları döndür
    return emotion, scores
# Kullanıcının duygu durumunu anlamak için ana fonksiyon
def understand_emotion():
    # Kullanıcıya sesli veya yazılı giriş seçeneği sun
    print("Botunuzla nasıl iletişim kurmak istersiniz?")
    print("1. Sesli")
    print("2. Yazılı")
    choice = input()
    # Kullanıcının seçimine göre giriş al
    if choice == "1":
        text = voice_to_text()
    elif choice == "2":
        text = get_text_input()
    else:
        print("Lütfen geçerli bir seçim yapın.")
        return
    # Girişin boş olup olmadığını kontrol et
    if text:
        # Girişi duygu durumu açısından analiz et
        emotion, scores = analyze_emotion(text)
        # Analiz sonucunu ekrana yazdır
        print("Duygu durumunuz: " + emotion)
        print("Puanlar: " + str(scores))
    else:
```

```
        print("Lütfen bir giriş yapın.")

        return
```

Bu kodlar, botumuzun emotion.py dosyasının içeriğini açıklamaktadır. Bu dosyada, kullanıcının duygu durumunu anlamak için gerekli kütüphaneler, modüller ve fonksiyonlar bulunmaktadır. Bu kodların ayrıntılı açıklamaları şöyledir:

- İlk kısımda, ses tanıma, API istekleri, metin konuşma, coğrafi konum hesaplamaları, veri analizi, doğal dil işleme ve rasgele seçim gibi işlevleri sağlayan kütüphaneler ve modüller içe aktarılır. Bu kütüphaneler ve modüller, botumuzun kullanıcının sesli veya yazılı girişlerini almasını, analiz etmesini ve cevap vermesini sağlar.

- İkinci kısımda, Borusan Otomotiv API'si için gerekli olan bağlantı bilgileri ve temel ayarlar tanımlanır. Bu bilgiler, botumuzun API'ye istek göndermesi ve cevap alması için gereklidir. Ayrıca, ses tanıma motoru ve ayarları da bu kısımda başlatılır.

- Üçüncü kısımda, `get_voice_input` fonksiyonu tanımlanır. Bu fonksiyon, kullanıcının sesli girişini alır ve Google Konuşma Tanıma API'sini kullanarak metne çevirir. Bu fonksiyon, kullanıcının sesli olarak botumuzla iletişim kurmasını sağlar.

- Dördüncü kısımda, `speak_response` fonksiyonu tanımlanır. Bu fonksiyon, botumuzun yanıtını ekrana yazdırır ve sesli olarak kullanıcıya iletilmesini sağlar. Bu fonksiyon, botumuzun kullanıcıya geri bildirim vermesini sağlar.

- Beşinci kısımda, `get_current_location` fonksiyonu tanımlanır. Bu fonksiyon, kullanıcının IP adresini kullanarak coğrafi konumunu alır. Bu fonksiyon, botumuzun kullanıcının konumuna göre bilgi vermesini sağlar.

- Altıncı kısımda, `get_nearest_gas_stations` fonksiyonu tanımlanır. Bu fonksiyon, kullanıcının konumuna en yakın akaryakıt istasyonlarını Borusan Otomotiv API'sini kullanarak bulur. Bu fonksiyon, botumuzun kullanıcıya yakıt konusunda yardımcı olmasını sağlar.

- Yedinci kısımda, `get_nearest_service_center` fonksiyonu tanımlanır. Bu fonksiyon, kullanıcının konumuna en yakın servis merkezini Borusan Otomotiv API'sini kullanarak bulur. Bu fonksiyon, botumuzun kullanıcıya araç bakımı konusunda yardımcı olmasını sağlar.

- Sekizinci kısımda, `explain_driving_data` fonksiyonu tanımlanır. Bu fonksiyon, sürüş performans verilerini değerlendirir ve bu verileri kullanıcıya anlaşılır bir şekilde açıklar. Bu fonksiyon, botumuzun kullanıcıya sürüş alışkanlıkları ve araç durumu hakkında bilgi vermesini sağlar.

- Dokuzuncu kısımda, `understand_emotion` fonksiyonu tanımlanır. Bu fonksiyon, kullanıcının duygu durumunu anlamak için sesli veya yazılı girişlerini analiz eder. Bu analizde doğal dil işleme (NLP) teknikleri kullanılır. Bu fonksiyon, botumuzun kullanıcının duygusal durumuna göre empati kurmasını ve uygun cevaplar vermesini sağlar.

Projenin Değerlendirilmesi

Projenin değerlendirilmesi aşamasında, projemizin sonuçlarını ve etkilerini analiz ettik. Bu analizde, projemizin AutoHack etkinliğinin amaçlarına ve Borusan Otomotiv'in beklentilerine ne kadar uygun olduğunu, projemizin otomotiv sektörüne ve topluma ne gibi katkılar sağlayabileceğini, projemizin sürdürülebilirliği ve yaygınlaştırılabilirliği gibi konuları ele aldık. Bu analiz sonucunda, projemizin başarılı bir şekilde tamamlandığını gösterdik.

Sonuç

Bu raporda, AutoHack için yapmış olduğumuz proje hakkında detaylı bir şekilde bilgi verdik. Projenin tasarımı, geliştirilmesi ve değerlendirilmesi aşamalarını anlattık. Projemizin, Borusan Otomotiv'in istediği özelliklere sahip bir otomotiv uygulaması olduğunu, kullanıcıların araçlarıyla iletişim kurmasını, araçlarının performansını ve durumunu takip etmesini, yakındaki akaryakıt istasyonları ve servis merkezleri gibi bilgilere erişmesini ve duygu durumlarına göre destek almasını sağladığını gösterdik. Projemizin, otomotiv sektörüne ve topluma faydalı bir katkı olduğunu kanıtladık. Projemizin, AutoHack etkinliğinin bir parçası olmasından dolayı çok mutluyuz ve projemizi daha da geliştirmek için çalışmaya devam edeceğiz. Teşekkürler.